Automated Incremental Design of Flexible Intrusion Detection Systems on FPGAs[1]

Zachary K. Baker and Viktor K. Prasanna
University of Southern California, Los Angeles, CA, USA
zbaker@halcyon.usc.edu, prasanna@ganges.usc.edu

Abstract

Intrusion detection for network security is a compute-intensive application demanding high system performance. This paper presents a variety of strategies we have developed for the automatic synthesis of highly efficient intrusion detection systems. We create FPGA architectures using a high-level, graph-based partitioning methodology. We provide a library of performance-customized architectures, which, through more efficient communication and extensive reuse of hardware components, provide dramatic increases in area-time performance. This paper addresses a problem of earlier designs, the requirement for complete place-and-route for small changes to the pattern database, through an optimized incremental design strategy.

1 Introduction

The continued discovery of programming errors in network-attached software has driven the introduction of increasingly powerful and devastating attacks [3, 7]. Attacks can cause destruction of data, clogging of network links, and future breaches in security. In order to prevent, or at least mitigate, these attacks, a network administratori can place a firewall or Intrusion Detection System at a network choke-point such as a company's connection to a trunk line (Figure 1). A firewall's function is to filter at the header level; if a connection is attempted to a disallowed port, such as FTP, the connection is refused. This catches many obvious attacks, but in order to detect more subtle attacks, an Intrusion Detection System (IDS) is utilized. The IDS differs from a firewall in that it goes beyond the header, actually searching the packet contents for various patterns that imply an attack is taking place, or that some disallowed content is being transferred across the network. Current IDS pattern databases reach into the thousands of patterns, providing for a difficult computational task.

[1]Supported by the United States National Science Foundation/ITR under award No. ACI-0325409 and in part by an equipment grant from the HP Corporation.

Methods commonly used to protect against security breaches include firewalls with filtering mechanisms to screen out obviously dangerous packets, and intrusion detection systems which use much more sophisticated rules and pattern matching to sense potential malicious packets. These techniques require significant computational resources. However, using automated design strategies for highly-parallel adaptive soft processors, there is potential for dramatic performance improvements. FPGAs provide an attractive platform for hardware implementation of intrusion detection because of the dynamic nature of the ruleset – as new vulnerabilities and attacks are identified, new rules must be added to the database and the device configuration must be regenerated.

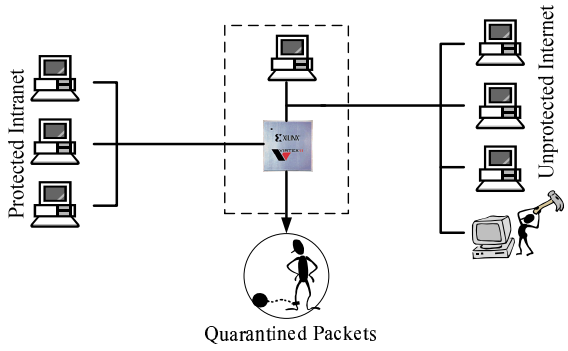


Figure 1. Intrusion detection systems protect networks from external threats. The use of FPGA allows a system to take advantage of massive parallelism in this is a highly computation-intensive task

This paper describes our work in creating Intrusion Detection Systems with *customized performance*, allowing a designer to mix and match from a collection of process steps and a family of architectures we have developed. Some of our results have already been published in [1].

Our basic architecture is a pre-decoded multiple-pipeline shift-and-compare matcher. While this ap-

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 FEB 2005	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Automated Incremental Design of Flexible Intrusion Detection Systems on FPGAs	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Southern California, Los Angeles,CA,USA**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM00001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004 Volume 1., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**18**	

proach can be considered "brute force" compared to a state machine approach [2, 4, 6] or a hashing approach [5], the simplicity of the units allows for exceptional area and time performance. The basic architecture, as described in detail below, reduces device routing and comparator size by converting incoming characters into many bit lines, each representing the presence of single character.

This basic architecture is extended in various ways. To allow for better area performance, we present a partial tree architecture that allows for significant reduction in redundant comparisons by independently matching prefixes that are shared across a range of patterns. To provide increased throughput performance, we provide a design that replicates a fraction of the hardware to allow for exact matching for k bytes per cycle. To provide high throughput with exceptional area efficiency, we provide an architecture that sacrifices exactness and allows for an increased false positive rate.

The architectures we have developed are only part of the contributions of this paper. To achieve better utilization of these architectures, system-level preprocessing steps are required, serving various functions including partitioning, grouping, and code generation. These steps, by considering the entire set of patterns in lieu of naïve hardware generation.

By intelligently processing an entire ruleset, our tool partitions the pattern collection into multiple pipelines in order to optimize the area and time characteristics of the system. The rule database is first converted into a graph representing the similarity of the ruleset. Depending on the flow, the graph edges are weighted to provide higher connectedness between rules with particular types of similar characters; this allows for increased grouping of prefixes as well as general shared-character grouping. The graph is partitioned based on the weighted graph and then sent to the partitioning routines, which act to reduce the interconnect burden in a given pipeline. Prefixes are then grouped for the tree architecture, if required. Based on this pre-processing, the system is generated from templates. By applying automated graph theory and trie techniques to the problem, the tool more effectively optimizes large ruleset as compared to naïve approaches.

2 Optimized Incremental Design

A problem with recent designs utilizing hard-wired comparator modules is in the requirement for a full place-and-route to make any change, no matter how small, to the design. Because of the exceptional area and time efficiency possible with this customized design paradigm, this issue has been largely ignored.

A portion of this paper covers our solution to the place-and-route problem. For the situation of adding a rule, we utilize the min-cut partitioned graph produced for the initial design. Determining the optimal partition to add a new pattern to is a fairly trivial task, only requiring a consideration of characters already mapped to the partition and pre-existing prefixes. The partition least modified by the addition of the new rule is determined by comparing the pre-decoded bits already within the partition, as well as the potential for using previously mapped prefixes. This VHDL code describing this partition is then modified by the tool. If the new pattern shares a prefix with some other pattern in the partition, the partial result of the previous pattern is mapped to the new pattern, reducing new wiring. The removal of rules is far easier, only the connections to the final result tree are removed. The new partition code is sent to the incremental synthesis and place-and-route functions of Xilinx ISE 6.2. The tool only re-synthesizes the modified modules. Because of the previously defined area constraints, each pipline module is independent of the others. Thus, only the routing in the modified module requires place and route.

Our initial results show that for a change of one pattern in a single partition in system with p partitions, the time for place-and-route is reduced to $1/p$ plus some overhead for reprocessing the guide files. This overhead can be fairly large (approaching 50% of the total PAR time). However,without the use of incremental place and route, the system would require a completely new place-and-route, or p times additional time.

References

[1] Z. K. Baker and V. K. Prasanna. A Methodology for the Synthesis of Effi cient Intrusion Detection Systems on FPGAs. In *The Twelfth Annual IEEE Symposium on Field Programmable Custom Computing Machines 2004 (FCCM '04)*, 2004.

[2] Z. K. Baker and V. K. Prasanna. Time and Area Effi - cient Pattern Matching on FPGAs. In *The Twelfth Annual ACM International Symposium on Field-Programmable Gate Arrays (FPGA '04)*, 2004.

[3] Z. Chen, L. Gao, and K. Kwiat. Modeling the Spread of Active Worms. *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, April 2003.

[4] C. R. Clark and D. E. Schimmel. Effi cient Reconfi gurable Logic Circuits for Matching Complex Network Intrusion Detection Patterns. In *Proceedings of FPL '03*, 2003.

[5] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood. Implementation of a Deep Packet Inspection Circuit using Parallel Bloom Filters in Reconfi gurable Hardware. In *Proceedings of HOTi '03*, 2003.

[6] B. L. Hutchings, R. Franklin, and D. Carver. Assisting Network Intrusion Detection with Reconfi gurable Hardware. In *Proceedings of FCCM '02*, 2002.

[7] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer Worm. *IEEE Security & Privacy Magazine*, 1(4), July-Aug 2003.

Introduction to Intrusion Detection Systems

Zachary K. Baker and Viktor K. Prasanna
University of Southern California
June 22, 2004

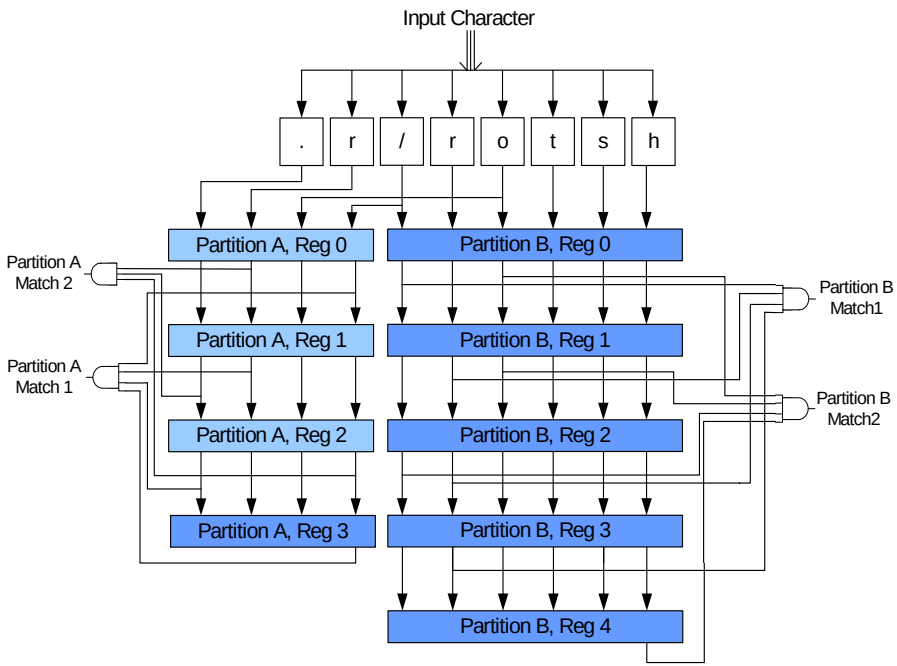
Introduction

- All incoming packets are filtered for specific characteristics or content,
- Databases have thousands of patterns requiring string matching

- We can achieve 10 Gb/s and higher rates desired
  - Provided by pipelined, streaming architectures,
  - Reduction of redundancy,
  - Efficient recoding,
  - Reduction of routing through pipeline partitioning

Efficient Matching Design

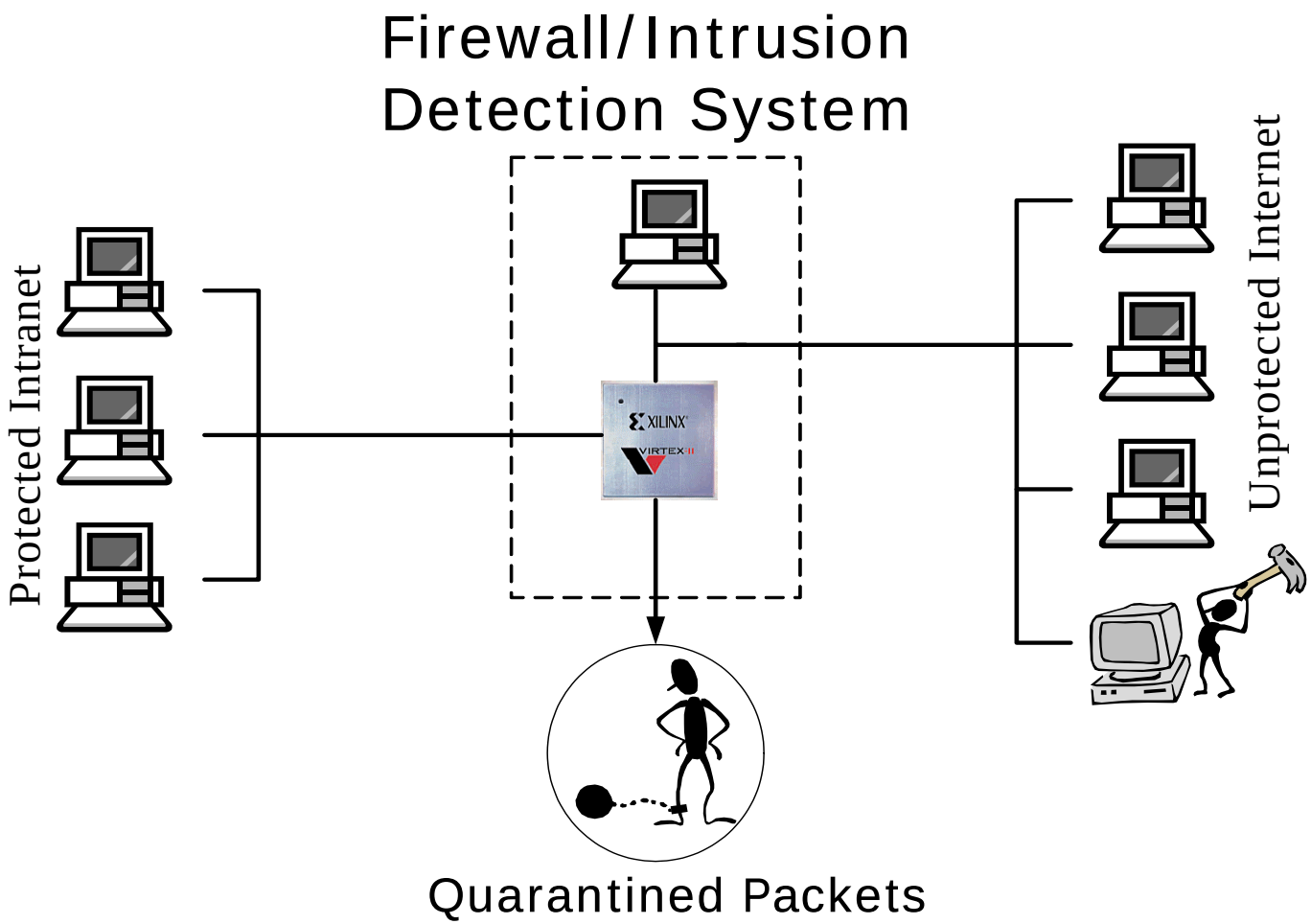
- Pre-decoding into individual characters allows for high time and area efficiency
- Pipelining allows for reduced interconnect latency and separation of related patterns into prefix-linked modules



Incremental Architecture Synthesis

- Module-based, partitioned pipelines allow for several independent modules connected only by controller
  - Changes in one module do not necessarily require recompilation of other modules
    - Significantly reduce place and route costs
    - Cost for changing rules in one of *k* partitions:
      overhead + 1/k

Introduction to Intrusion Detection Systems



What is Intrusion Detection?

- All incoming packets are filtered for specific characteristics or content
- Databases have thousands of patterns requiring string matching
  - FPGA allows fine-grained parallelism and computational reuse
- 10 Gb/s and higher rates desired
  - Provided by pipelined, streaming architectures

Other Approaches

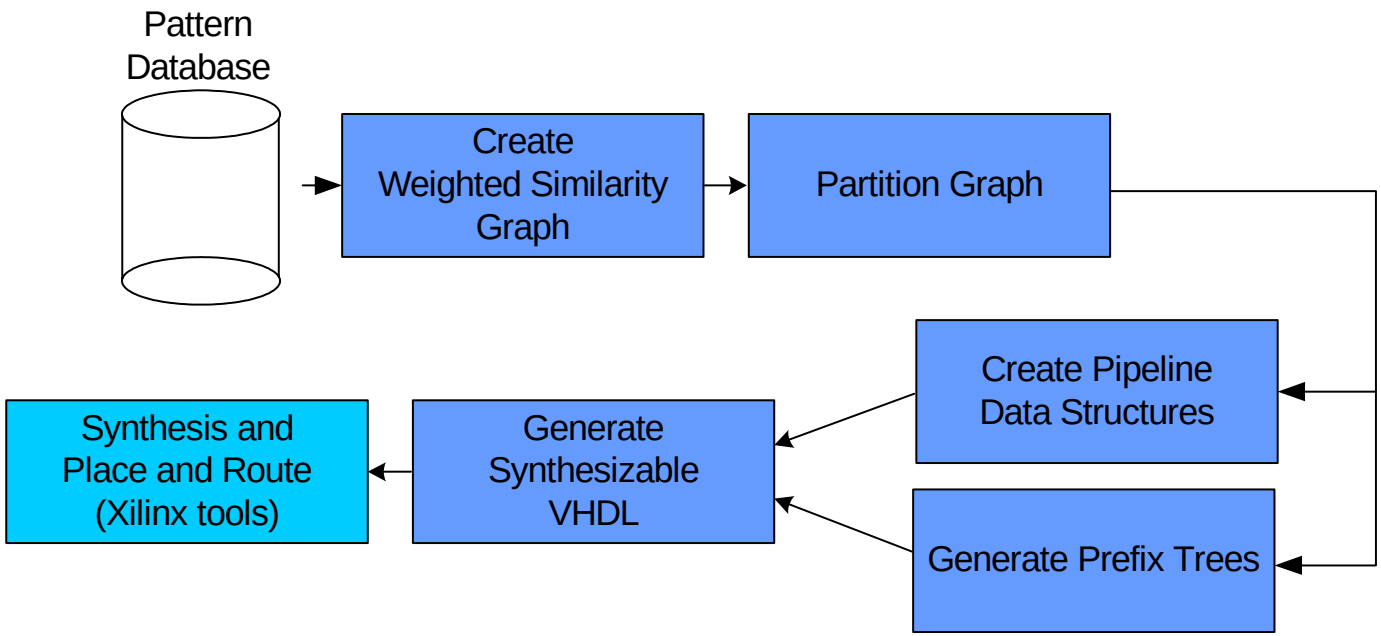
- Objective: find all occurrences of a pattern in an input
- Naïve approach: O((n-m+1)m)
- Shift-and-compare: O(n), large hardware requirements, O(nm) work
- Hashing: O(n), hashing can be complex, O(nm) work
- KMP: O(n): other algorithms may be faster in practice, but do not provide low precise upper bound (2n – m), O(n+m) work

High-Performance Shift-and-Compare Architectures

Various contributions to shift-and-compare architectures:

- Pre-decoded architecture provides significant area and routing improvements over encoded data
- Graph-based partitioning of patterns allows for reduced routing complexity and increased frequency performance through multiple pipelines
    - Average of 15% decrease in area, 5% decrease in clock period over unpartitioned unary

Methodology Flow

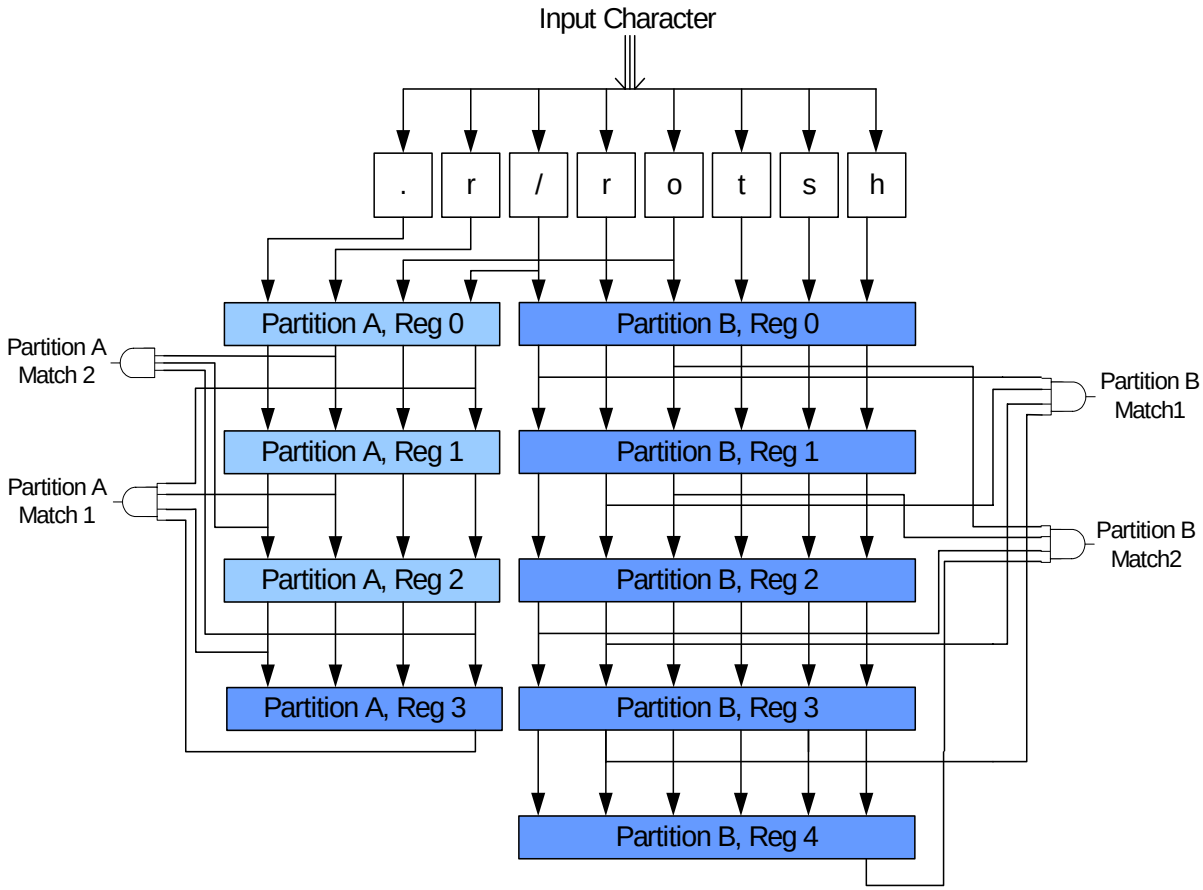


Reduction of Resource Usage

- Trie-based prefix grouping allows for reduced area consumption through lower redundant comparisons
  - 4-byte prefixes turn out to be very appropriate for intrusion detection:

    /cgi-bin/bigconf.cgi
    /cgi-bin/common/listrec.pl
    /cgi-sys/addalink.cgi
    /cgi-sys/entropysearch.cgi

- Replication of hardware and delays allow for multi-byte per cycle throughput at high clock rates
  - Pipeline is not increased in size – large source of slice consumption
  - Front end decoders increases in size by *k*
  - Back end matchers increase in size by *k*



	1 way	4 way	8 way
Number of Slices	299	721	1338
Clock Period	4.2ns	4.6ns	5.3ns
Throughput	1.9Gb/s	6.9Gb/s	12.1Gb/s
Efficiency	1	1.51	1.41

*Efficiency in throughput/area, normalized to 1-way (~100 rules)

Customized Performance

- Variations in tool flow provide customizable performance:
  - Tool Options
    - Small: partitioned and pre-decoded architecture
      - Prefix trees
    - Fast: *k*-way architecture
    - Fast reconfiguration, minimum complexity
      - KMP architecture

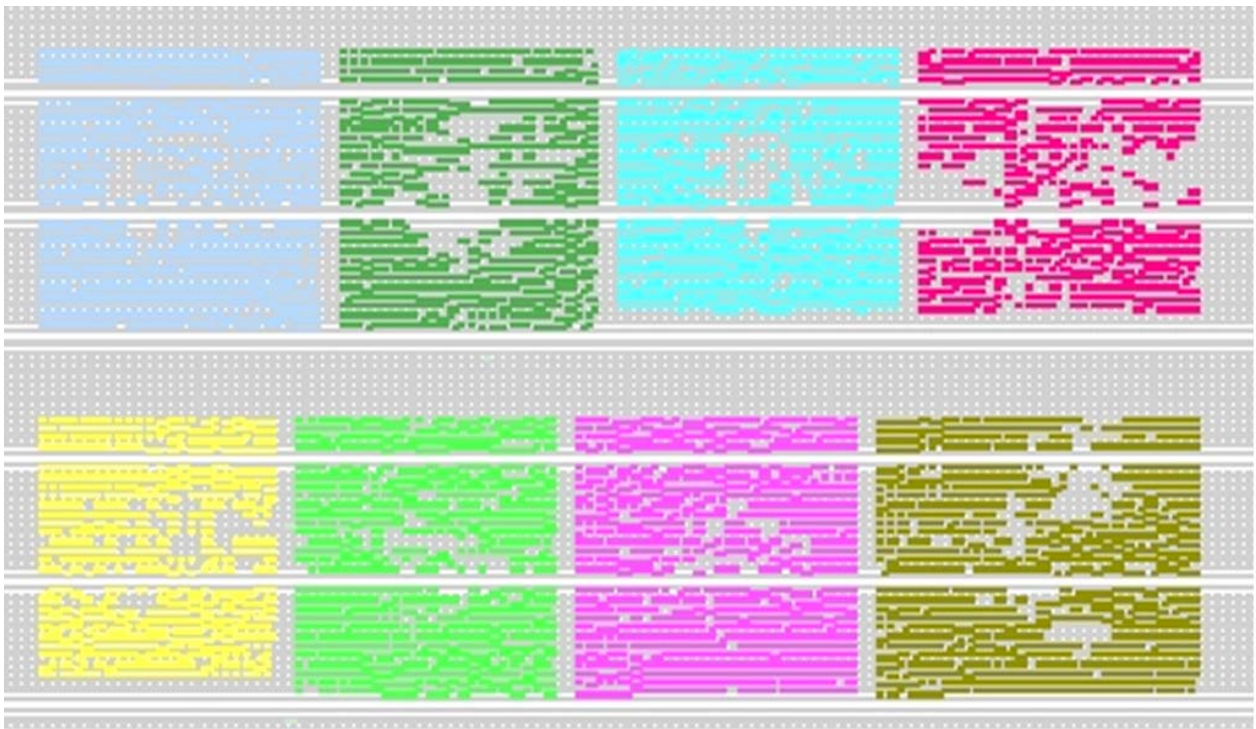
Comparison of Related Architectures

Design	Throughput	Unit Size	Performance
USC Unary	2.1 Gb/s	7.3	283
USC Unary (1 byte)	1.8 Gb/s	5.7	315
USC Unary (4 byte)	6.1 Gb/s	22.3	271
USC Unary (8 byte)	10.3 Gb/s	32	322
USC Unary (Prefilter)	6.4 Gb/s	9.4	682
USC Unary (Tree)	2.0 Gb/s	6.6	303
Los Alamos (FPL '03)	2.2 Gb/s	243	9.1
UCLA (FPL '02)	2.9 Gb/s	160	18
UCLA w/Reuse (FCCM '04)	3.2 Gb/s	11.4	280
U/Crete (FPL '03)	10.8 Gb/s	269	40.1
U/Crete (FCCM '04)	9.7 Gb/s	57	170
GATech (FCCM '04)	7.0 Gb/s	50	140

* Throughput is assumed to be constant over variations in pattern size. Unit size is the average unit size for a 16 character pattern (in logic cells; one slice is two logic cells), and performance is defined as Mb/s/cell).

Incremental Architecture Synthesis

- Goal: Reduce place and route costs
- Cost for changing rules in one of *k* partitions:
  overhead + 1/k
- Key: Predefinition of area constraints



Determining the Optimal Partition

$$\delta_i = \left(S_{p^*} \setminus P_i\right)$$

$$\text{find j such that } \left|\delta_j\right| = \min_{i=0}^{P} \left|\delta_i\right|$$

$$\text{characters to add to partition j are in } \delta_j$$

Definitions:

- S_{p^*} the set of characters required to represent the new pattern p*.
- The set difference between the characters currently represented in P_i and the characters that are present in S_{p^*} is δ_j
- The partition which will require the addition of the minimum number of new characters is the optimal partition P_j.
- The optimal partition is selected from the set of partitions P.

Relevant Publications

"*Time and Area Efficient Pattern Matching on FPGAs*," Proceedings of the 12th Annual ACM International Symposium on Field-Programmable Gate Arrays (FPGA '04)

"*A Methodology for the Synthesis of Efficient Intrusion Detection Systems on FPGAs*," Proceedings of the Twelfth Annual IEEE Symposium on Field Programmable Custom Computing Machines 2004 (FCCM '04)

"*Automated Incremental Design of Flexible Intrusion Detection Systems on FPGAs*," Proceedings of the Eighth Annual Workshop on High Performance Embedded Computing (HPEC '04)

Additional publications: http://ceng.usc.edu/~prasanna